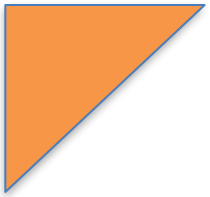


Lecture L6
Performance Enhancement

Lecture 6-01
Caches

Clive Maynard

clive.maynard@monash.edu

COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

WARNING

This material has been reproduced and communicated to you by or on behalf of Monash University pursuant to Part VB of the Copyright Act 1968 (the Act).

The material in this communication may be subject to copyright under the Act. Any further reproduction or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice.

L6 Performance Enhancement

Improving computer performance in terms of processing speed has been a constant objective of computer designers.

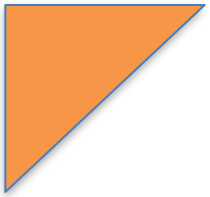
An obvious way of improving computer throughput is to increase the system clock speed. This in turn requires faster logic and memory devices. The limits here are the state of current technology and cost.

In this section we will look at two commonly used techniques for increasing the throughput of a computer. These are:

- the use of memory caches and
- processor pipelining.

A third effective technique is multi-core which is not covered in this unit

Marilyn Wolf, Computers as Components, 4th Edition., Morgan Kaufmann



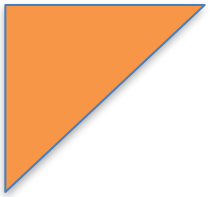
Memory Access Time

Questions:

1)If the main memory of a computer has an access time of x nanoseconds what is the average memory access time?

2)The main memory of a computer is made up of 90% slow memory with an access time of x nanoseconds and 10% fast memory with access time of x/10 nanoseconds. If accesses to memory are performed at random what is the average memory access time?

3)When running a computer program with program code and data stored in memory would you expect memory accesses to be performed on random locations?



Memory Caches

A cache is a relatively small amount of fast memory that holds copies of some of the contents of main memory.

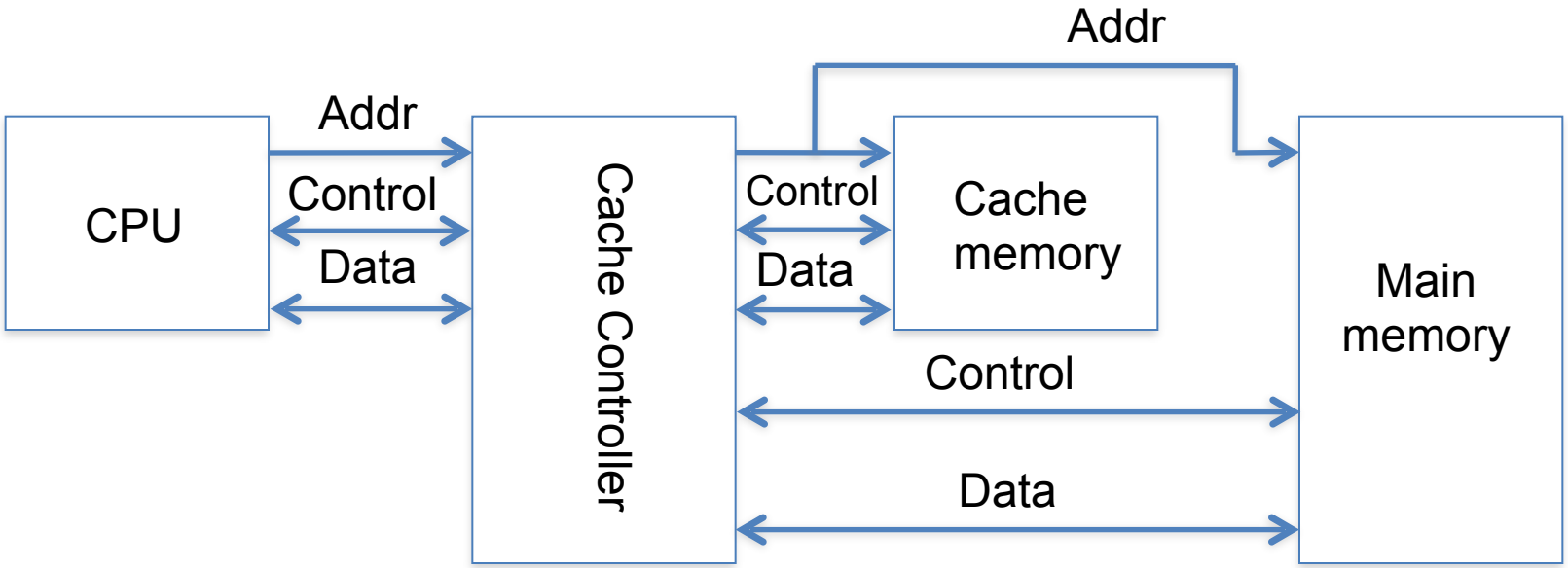
By relying on spatial coherence (spatial locality) and temporal coherence (temporal locality) in memory accesses the cache can be arranged to contain the information required by the processor most of the time. Thus speeding up memory accesses.

The range of memory locations used by a program is called the Working Set.

• spatial coherence = if the processor accessed a particular memory location the chances are it will access an adjacent memory location soon.

• temporal coherence = if the processor accessed a particular memory location the chances are it will access the same memory location again soon.

Cache Read



CPU requests memory contents from a particular address

If the cache controller finds that requested data is in the cache (a cache hit) then the cached value is passed to CPU

Otherwise cache controller requests data from main memory (a cache miss)

Average Memory Access Time

$t_{av} = h * t_{cache} + (1 – h)\ t_{main}$

Where:

t_{av} = average memory access time

h = hit rate

t_{cache} = cache access time

t_{main} = main memory access time

Average Memory Access Time

If we require an overall average access time of 8ns in a single level cache system with main memory access time of 60ns and cache access time of 4ns what cache hit rate is required?

$t_{av} = h * t_{cache} + (1 - h)\, t_{main}$

Where:

t_{av} = average memory access time

h = hit rate

t_{cache} = cache access time

t_{main} = main memory access time

$$8 = 4h + (1-h)60$$

$$8 = 4h + 60 - 60h$$

$$56h = 52$$

$$h = 0.93$$

Two-Level Cache System

It is common to have more than one level of cache memory. The first level (L1) is closest to the cpu, second level (L2) feeds into the first level and so on ….

$$t_{av} = h_1 * t_{L1} + h_2 * t_{L2} + (1 - h_1 - h_2)\, t_{main}$$

Note that h_2 is the hit rate for second level cache but not the first level cache. That is $h_2 = (1 - h_1)\, h_2'$ where h_2' is the hit rate of the level 2 cache on its own.

Example

Assume that a system has a two-level cache:
The level I cache has a hit rate of 90% and the level 2 a hit rate of 97%. The level 1 cache access time is 4ns, the level 2 access time is 25ns and the main memory access time is 80ns. What is the average memory access time?

$$t_{av} = h_1 * t_{L1} + h_2 * t_{L2} + (1 - h_1 - h_2)\, t_{main}$$

In the question the level 2 hit rate (97%) is the level 2 cache hit rate on its own.

h2 = (1-0.9)*0.97
h2 = 0.097
tav = 0.9*4 + 0.097*25 + (1-0.9-0.097)*80
tav = 3.6 + 2.35 + 0.24 = 6.19

Memory Caches

The BIG picture:

A cache is a relatively small amount of fast memory that holds copies of some of the contents of main memory.

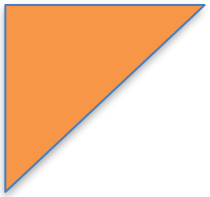
By relying on spatial coherence (spatial locality) and temporal coherence (temporal locality) in memory accesses the cache can be arranged to contain the information required by the processor most of the time. Thus speeding up memory accesses.

How does it work?

A wonderful idea but how can it be implemented in practice? A lot goes on behind the scenes to make a cache work.
In this section the basics of cache operations are described.

More information can be found in Patterson and Hennessy and many other sources.

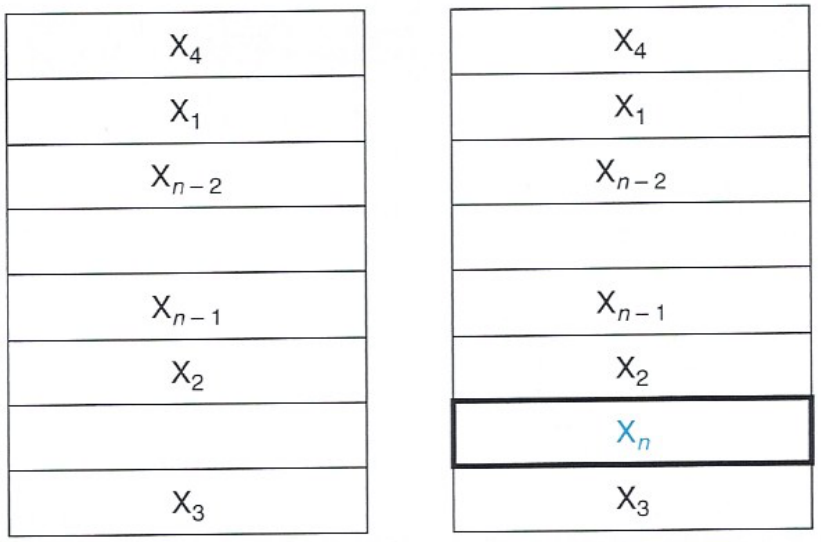
clive.maynard@monash.edu



Memory Caches

- What is the critical relationship between successful use of a cache and the working set?
- If the working set is larger than the cache size then the cache is continually being updated from the main memory.
- If the working set is smaller than the cache size then after the preliminary loading of the cache all references are found in the cache.

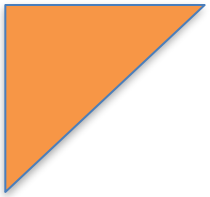
The cache before and after reference to a word x_n that is not initially in the cache.



a. Before the reference to X_n b. After the reference to X_n

Two questions?
How do we know if the data item is in the cache?
If it is, how do we find it?

clive.maynard@monash.edu



Where is the data?

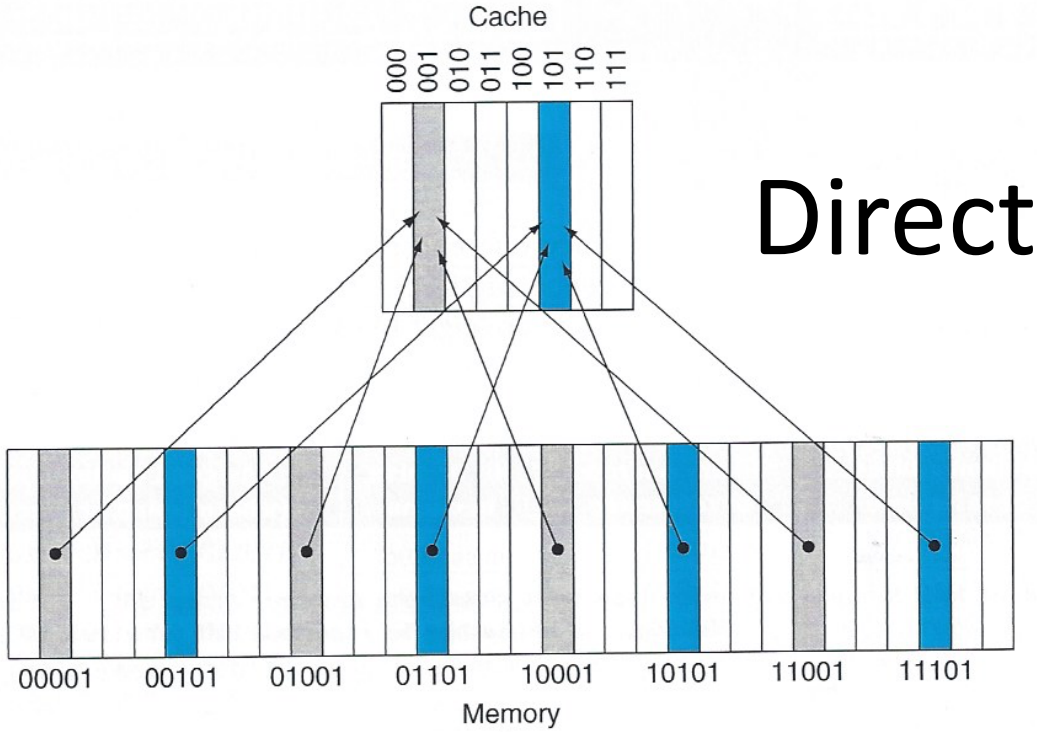
If each word can only go into exactly one place in the cache, then it is straightforward to find the word if it is in the cache.
The simplest way is to assign the location based on the address of the word in memory.
This is called **Direct Mapped.**

Almost all use this mapping to find a block:
(Block address) modulo (Number of blocks in the cache)

If the number of entries in the block is a power of 2 we can use the low order address bits of the address.
An 8-block cache would use the 3 lowest order bits of the address

clive.maynard@monash.edu

Direct mapped cache.



As a number of memory locations map to the same location in the cache, how do we know whether the the data in the cache is from the requested memory word?

We introduce a set of **tags**. These tags identify the memory source for the cache data. They contain the higher order address bits not used in the index into the cache.

In the example the upper two bits of the memory address provide the tag to identify which memory locations are copied into the cache.

Is it really in the cache? A **valid bit** needs to be introduced to ensure this is known.

An example follows (apologies for the slightly tipped images!!!)

clive.maynard@monash.edu

An example operation of a cache.

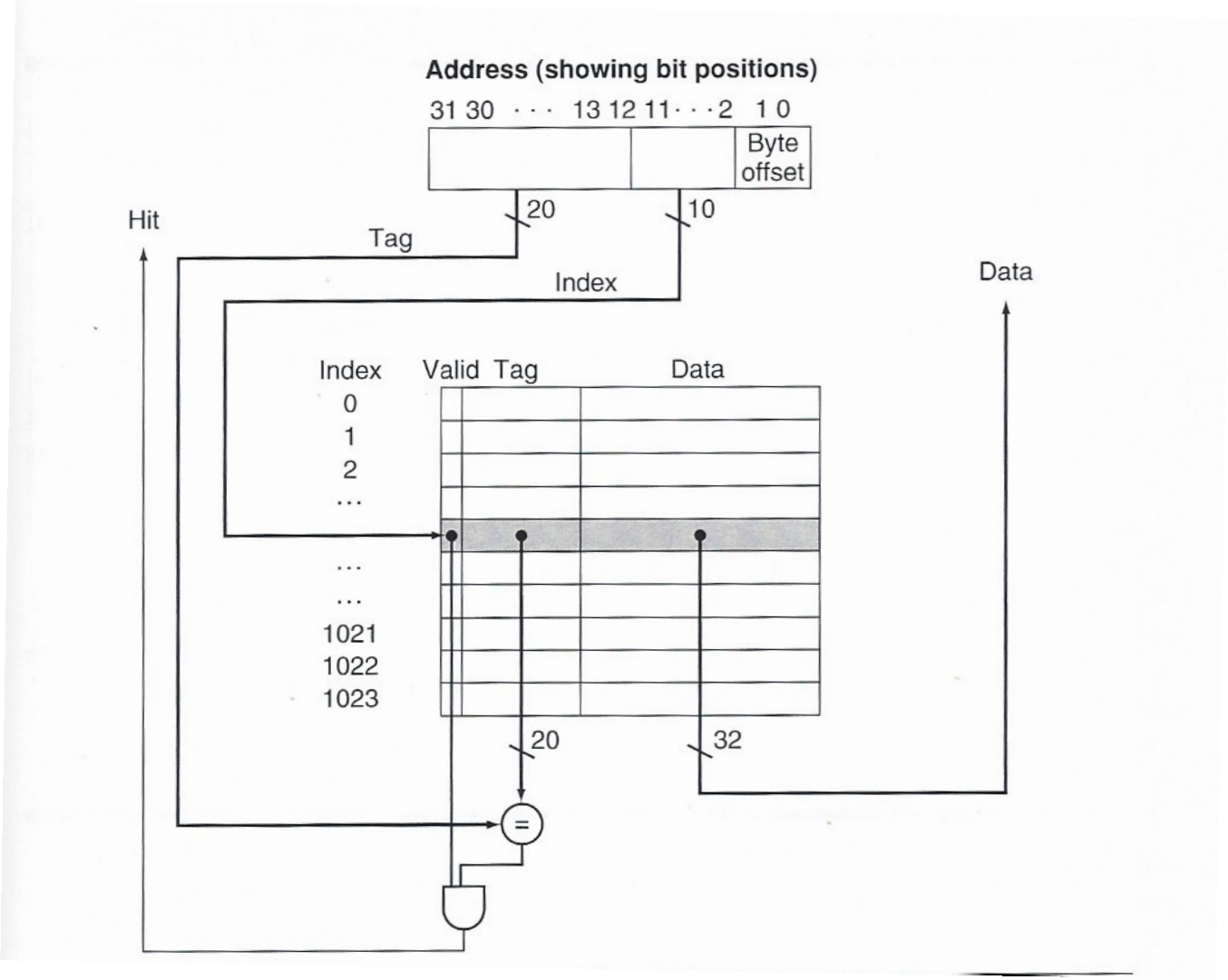
Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss (5.9b)	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	miss (5.9c)	$(11010_{two} \bmod 8) = 010_{two}$
22	10110_{two}	hit	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	hit	$(11010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	miss (5.9d)	$(10000_{two} \bmod 8) = 000_{two}$
3	00011_{two}	miss (5.9e)	$(00011_{two} \bmod 8) = 011_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$
18	10010_{two}	miss (5.9f)	$(10010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	10_{two}	Memory (10010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

f. After handling a miss of address (10010_{two})

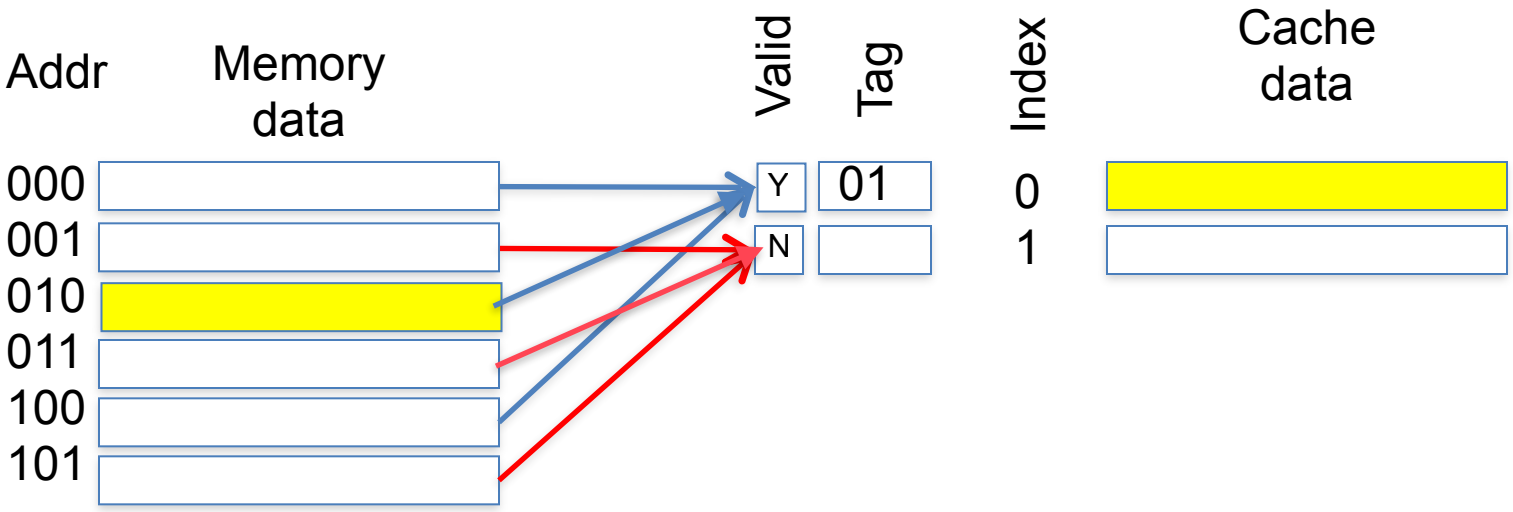
clive.maynard@monash.edu

An Implementation

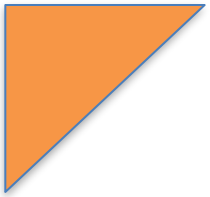


clive.maynard@monash.edu

Direct Mapped Cache Read



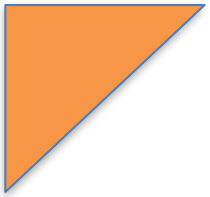
1) Find the cache entry with index bit/s corresponding to the associated bit/s in the memory address.
2) If the tag field matches the corresponding bits in the memory address and the valid bit is set then the cache contains the required data (a cache hit)



Cache Read Misses

If the required memory data is not available in the cache then the CPU has to be put into a wait condition.

The data is then requested from main memory and when it arrives it is placed in cache (possibly replacing an earlier entry) and passed on to the CPU which can then continue.

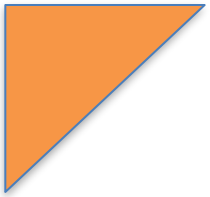


Classify Cache Misses

• *Compulsory misses* are those misses caused by the first reference to a memory location (sometimes referred to as *cold start misses)*.

• *Capacity misses* are those misses due to the finite size of the cache (the whole of the working set is too big to fit in the cache).

• *Conflict misses* are those misses that could have been avoided, had the cache not evicted an entry earlier.
Conflict misses can be further broken down into
mapping misses, that are unavoidable given a particular amount of associativity,
and
replacement misses, which are due to the particular choice of the replacement policy.



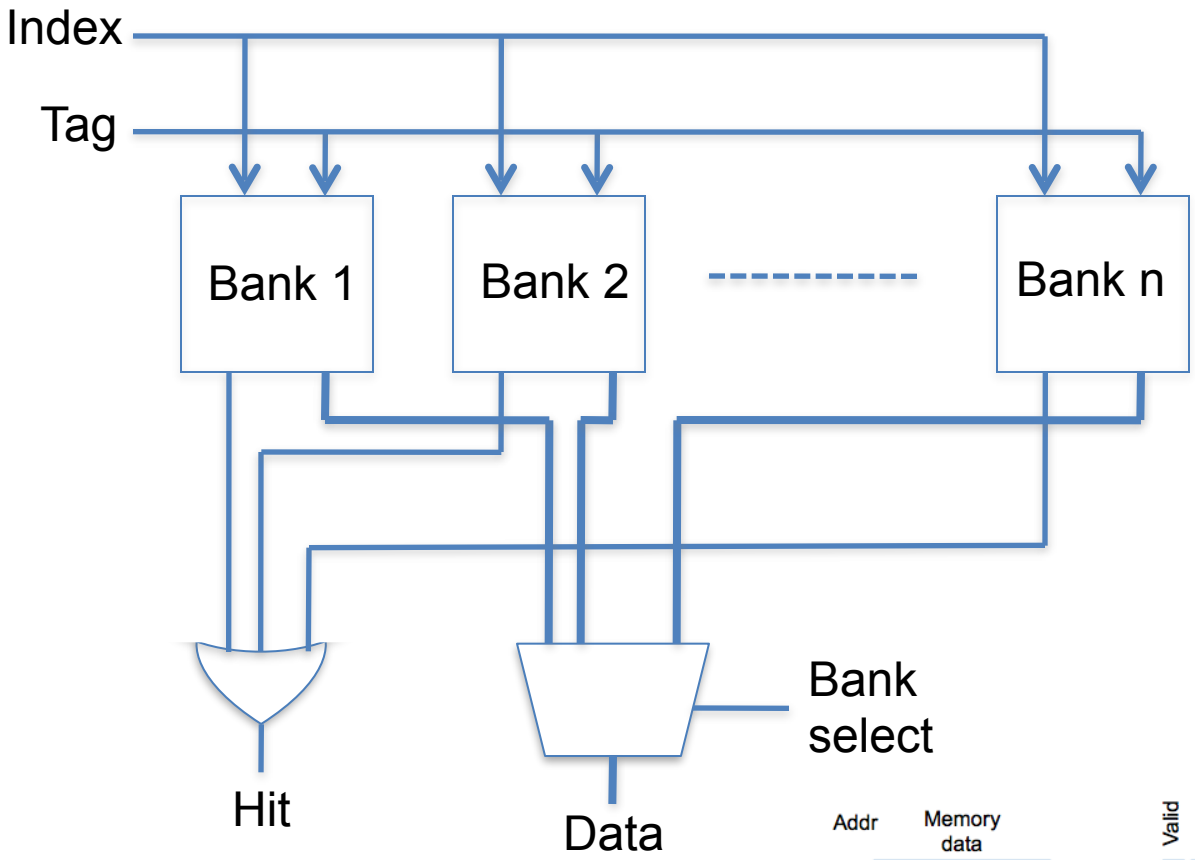
Direct Mapped Cache Write

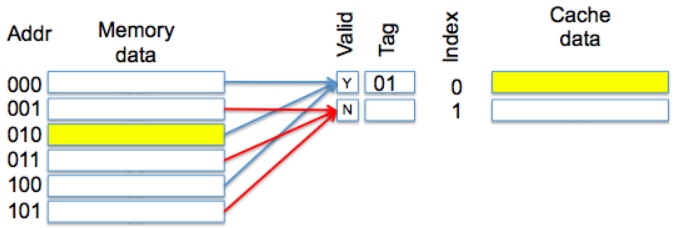
Question: when running an average program what would be the proportion of memory reads compared to memory writes?

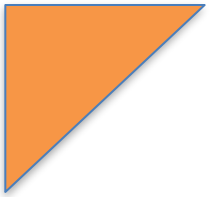
Write-through – every time the computer performs a memory write it updates both the cache and memory at the same time.

Using a write-back policy – such policies may be only write the cache value back to main memory when the entry is removed from cache.

Set-Associative Caches







Set-Associative Caches

A limitation of the direct-mapped cache is that there is only one storage location for each index value. The index value represents the least significant bits (LSBs) of the memory address. Therefore, if the computer accesses two or more regions of memory where these LSBs are the same the cache will regularly experience misses.

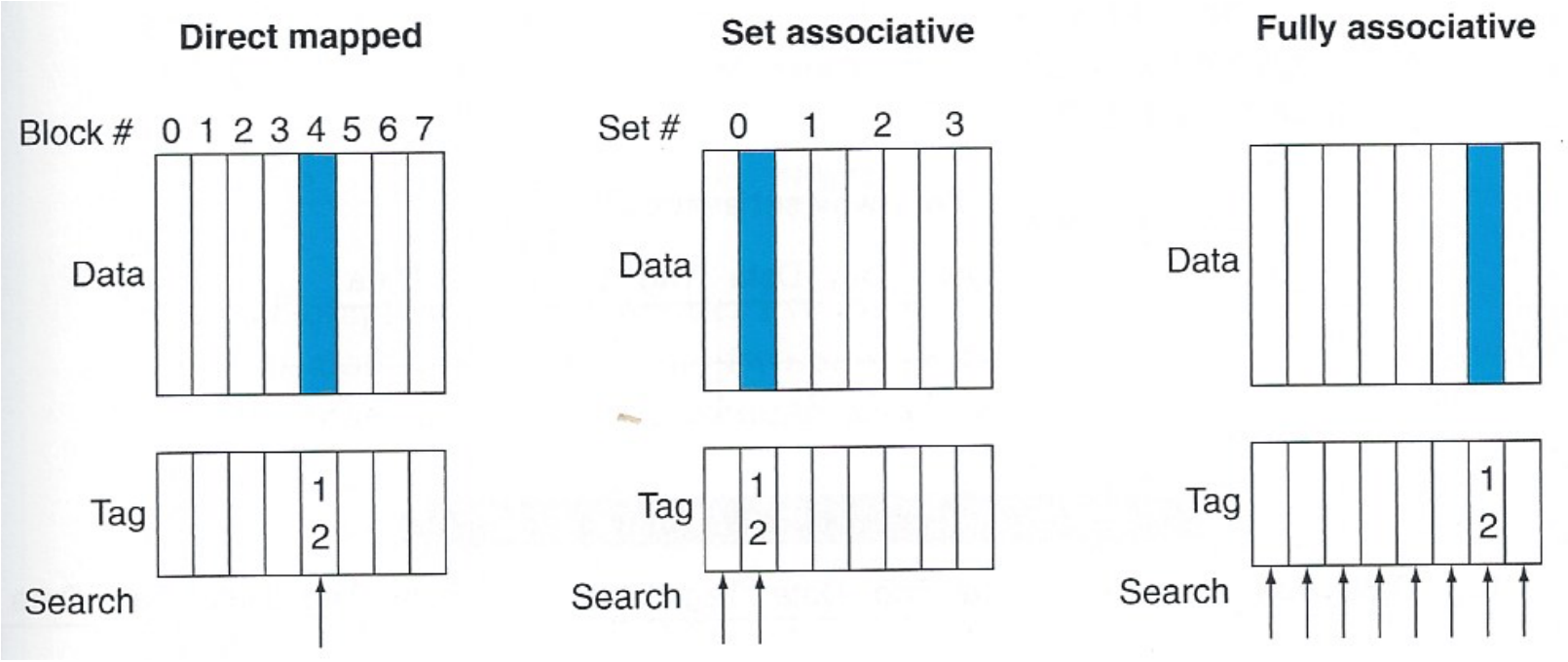
Set-associative caches attempt to get over this problem by utilizing n banks of data registers. Each bank contains a set of data registers addressed by the index value. Therefore, for one particular index value there will be one corresponding data register in each of the n banks. This group of n registers with the same index is called a set.

Therefore, for any particular index value there are n possible storage locations. If a cache miss occurs then one of the n possible locations is selected for replacement – one possible policy is least-recently-used replacement.

INVESTIGATE

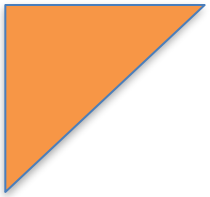
How do you find out which is the least recently used location?

Improving cache performance



In Direct Mapped there is only one cache block where memory block 12 can be found.
12 modulo 8 = 4
In two-way set associative there are 4 sets and memory block 12 must be in set
12 modulo 4 = 0 and located in either element of the set.
In fully associative the memory block can appear in any of the eight cache blocks.

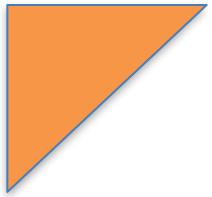
clive.maynard@monash.edu



Lecture 6-02
Pipelining

Clive Maynard

clive.maynard@monash.edu

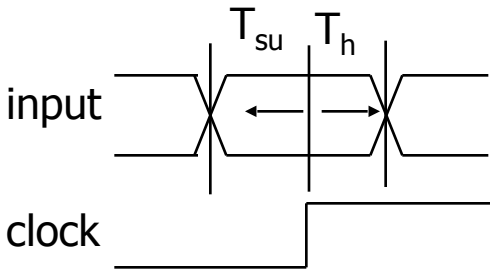


Improving data throughput

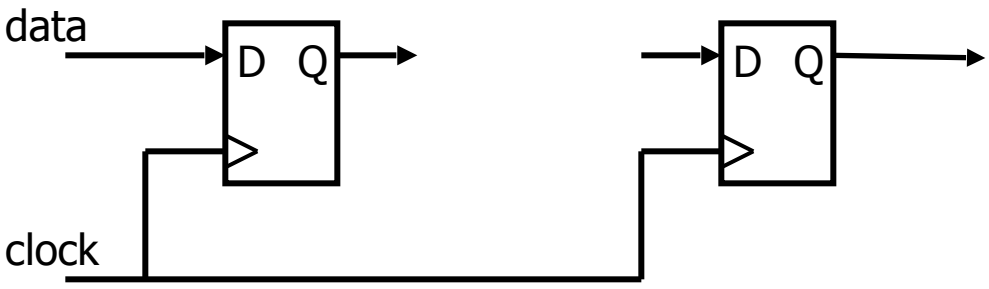
Now we will look at a method of increasing the number of instructions that a microprocessor can execute per second.

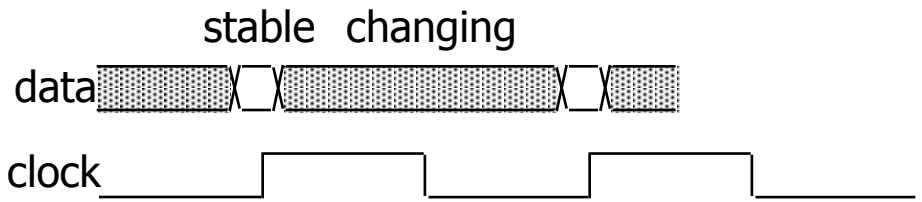
You will Recall

- Definition of terms
  - clock: periodic event, causes state of memory element to change
    can be rising edge or falling edge or high level or low level
  - setup time: minimum time before the clocking event by which the input must be stable (Tsu)
  - hold time: minimum time after the clocking event until which the input must remain stable (Th)

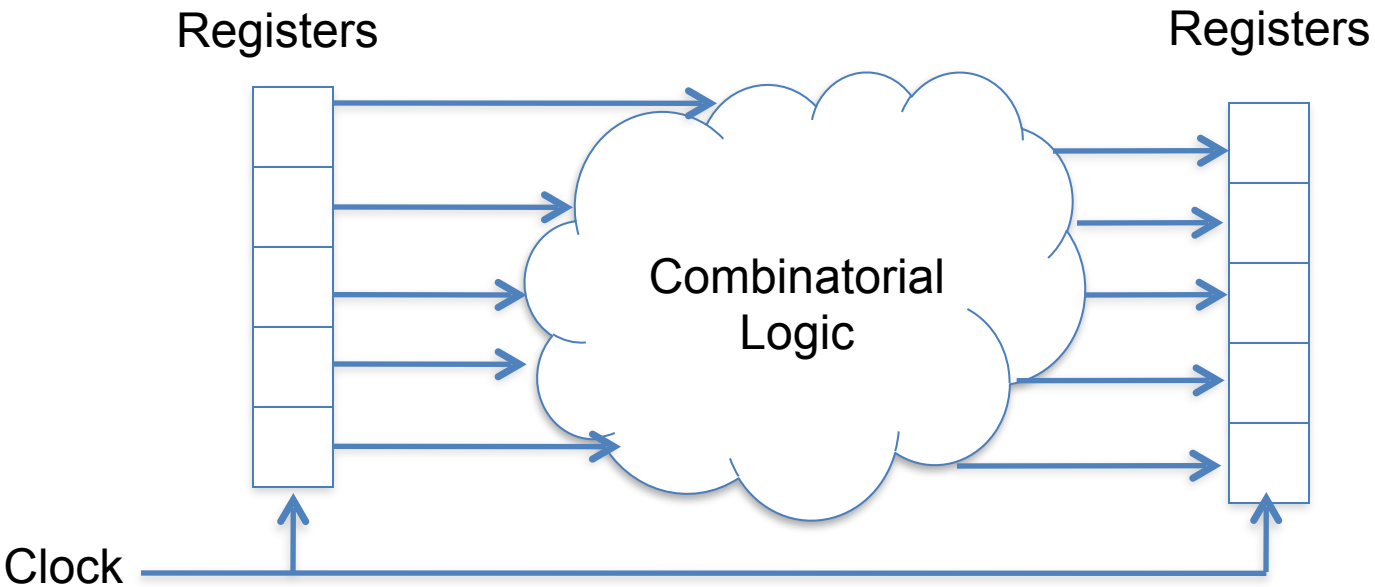


there is a timing "window" around the clocking event during which the input must remain stable and unchanged in order to be recognised



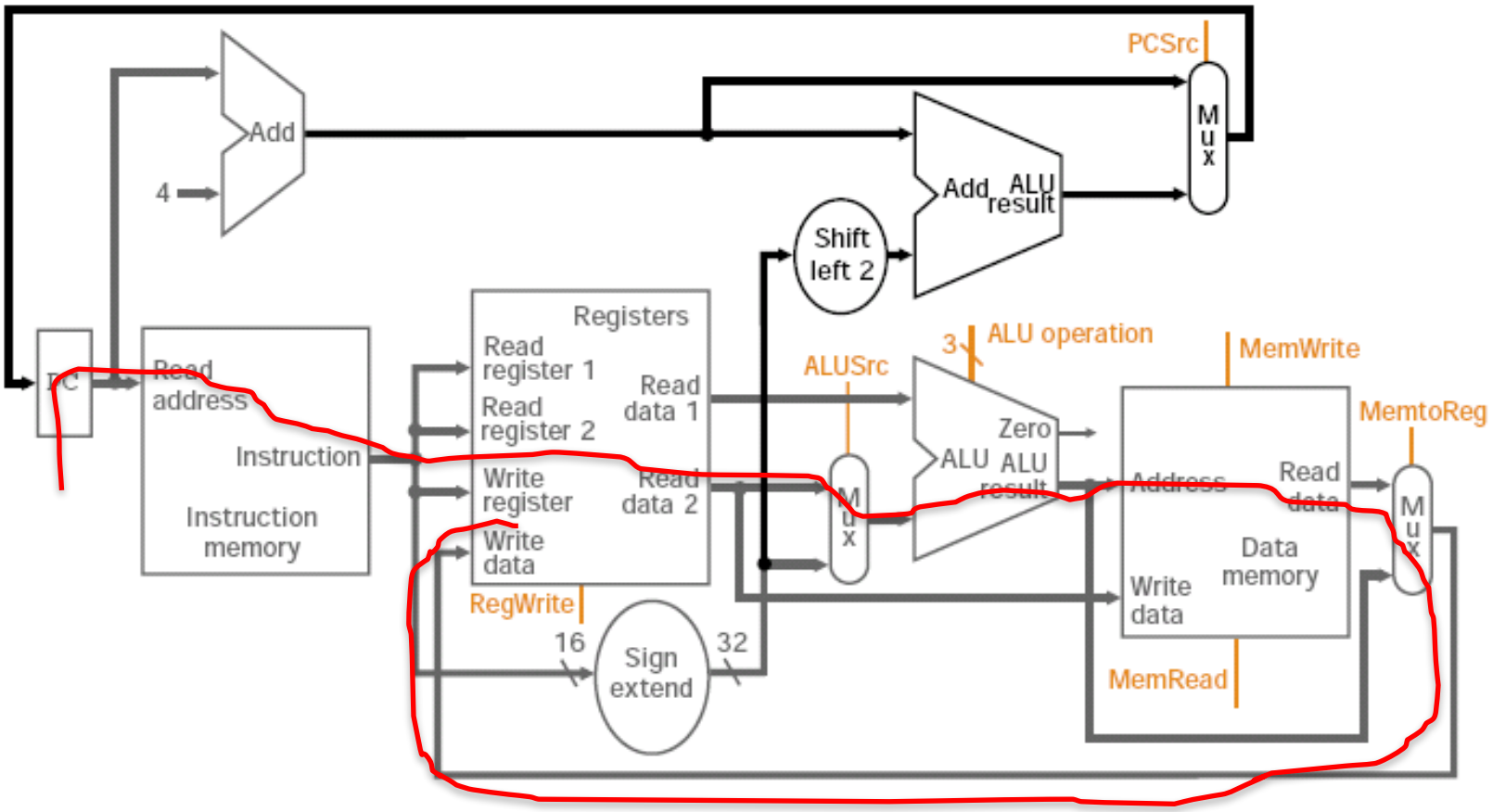


And



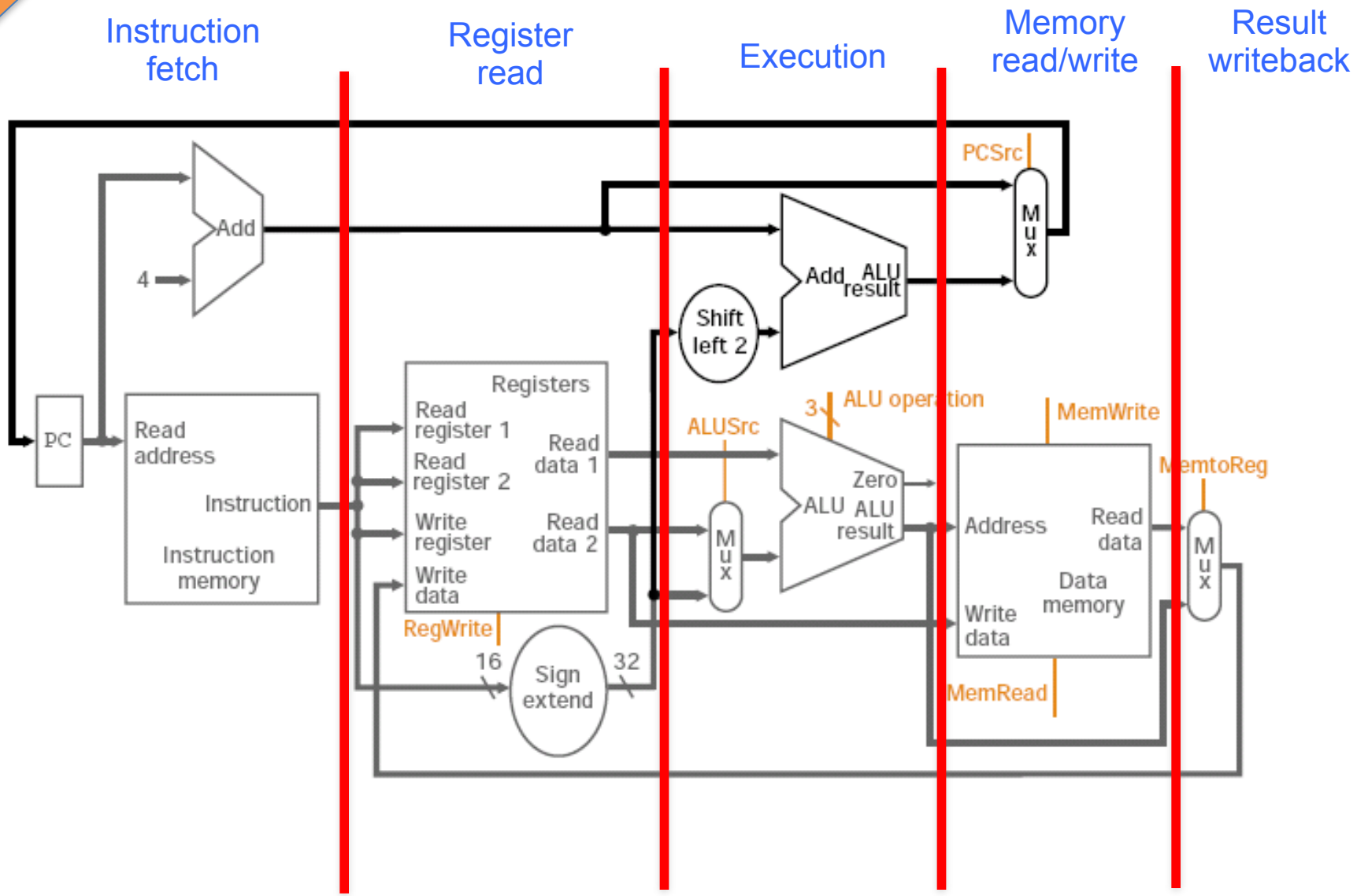
Also

MIPS Data Path

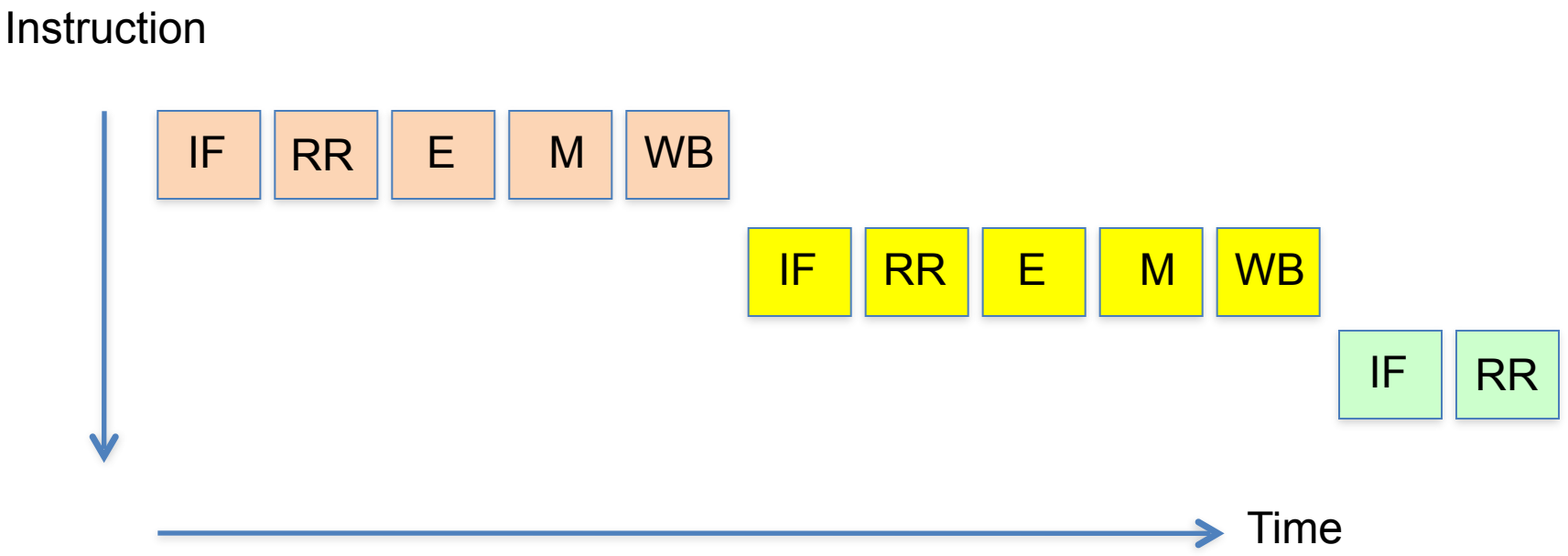


https://youtu.be/YGSAWqQy9bI

Pipelining

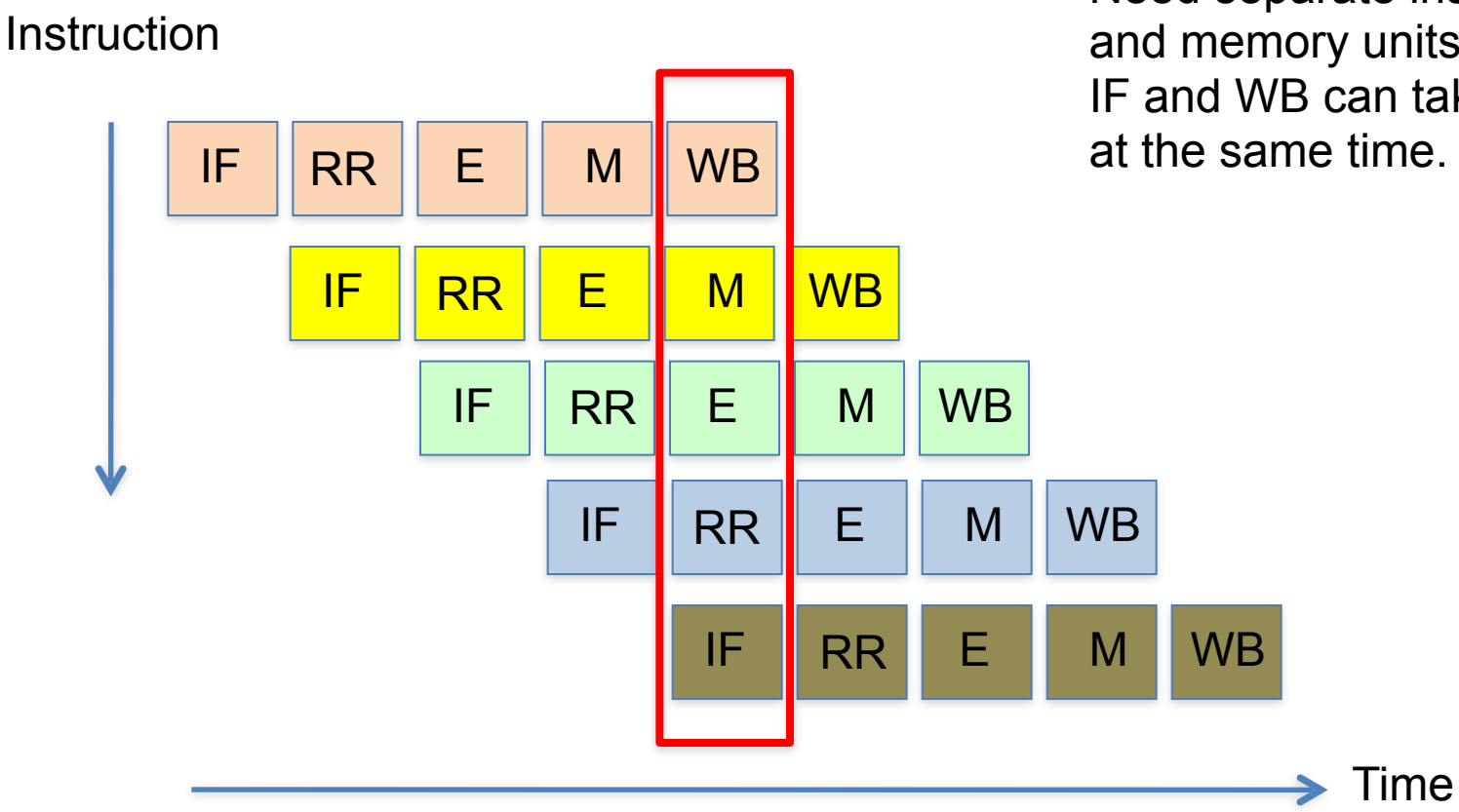

Instruction fetch
Register read
Execution
Memory read/write
Result writeback
Add
4
PCSrc
M u x
Add
ALU result
Shift left 2
PC
Read address
Instruction
Instruction memory
Registers
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
RegWrite
16
Sign extend
32
ALUSrc
3
ALU operation
M u x
Zero
ALU
ALU result
MemWrite
MemtoReg
Address
Read data
Data memory
Write data
MemRead
M u x

Pipelining (not)

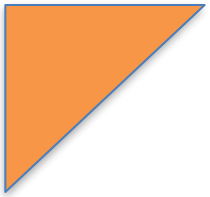


IF = Instruction Fetch
RR = Register Read
E = Execute
M = Memory Read/Write
WB = Write Back result

Pipelining



Need separate instruction and memory units so that IF and WB can take place at the same time.



Pipelining

Pipelining improves performance by increasing throughput as opposed to decreasing execution time of each individual instruction.

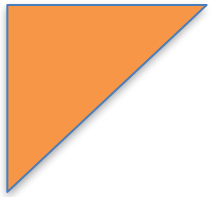
Like having a larger diameter pipe with the same flow rate to transport more water.

Timing

Instruction class	Instruction Fetch	Register read	Execution	Memory read/write	Result writeback	Total time
Load word (lw)	2ns	1ns	2ns	2ns	1ns	8ns
Store word (sw)	2ns	1ns	2ns	2ns		7ns
R-format (add, sub, and, or, slt)	2ns	1ns	2ns		1ns	6ns
Branch (beq)	2ns	1ns	2ns			5ns

For non-pipelined architecture instruction cycle time must be longer than total delay time through all stages = 8ns

For pipelined architecture average time between instructions is longest time through a stage = 2ns



Timing Improvement

In a pipelined processor one new instruction will be started after each stage delay.

Therefore, for a long sequence of *m* instructions processed on a pipelined processor with n pipeline stages then rather than taking

n*m*stage_delay seconds

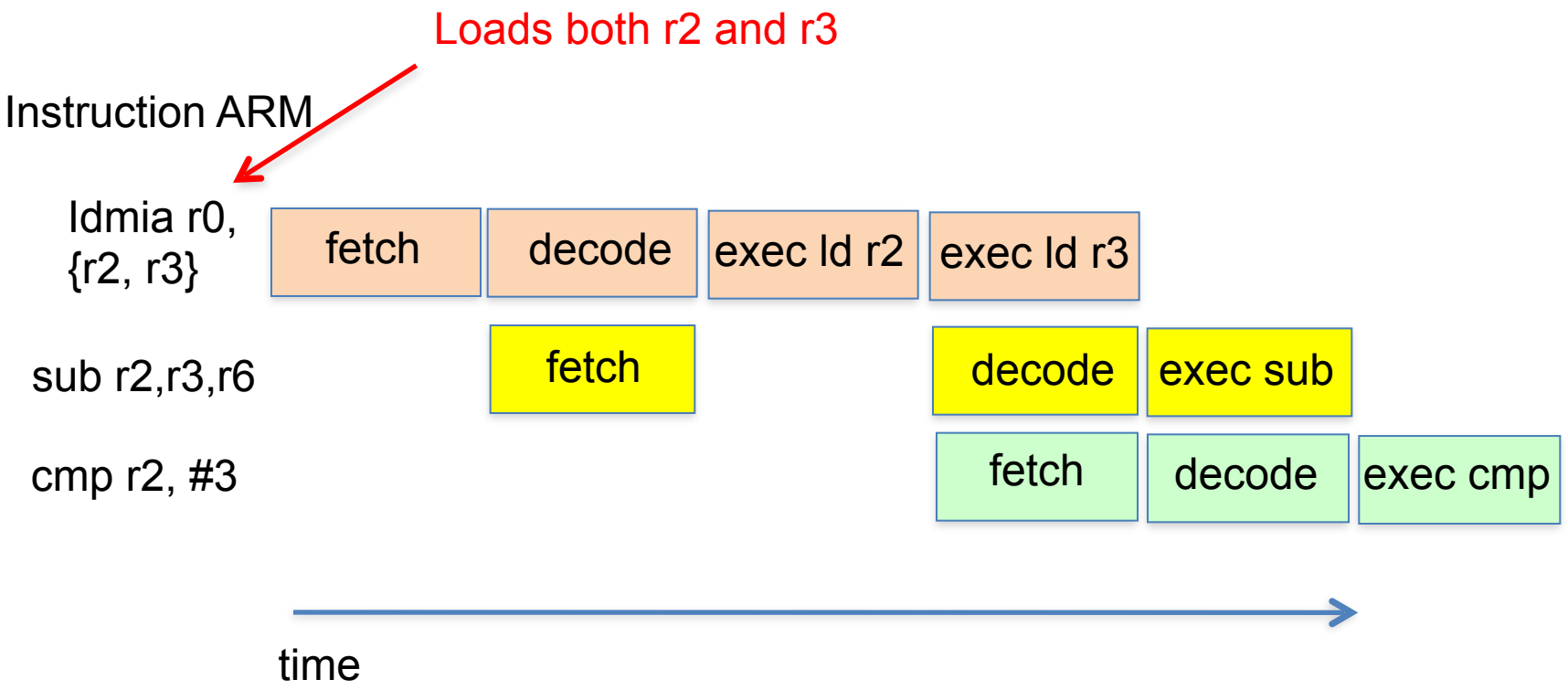
To execute the instructions the time will be approximately

m*stage_delay seconds

Representing a speedup of n times!

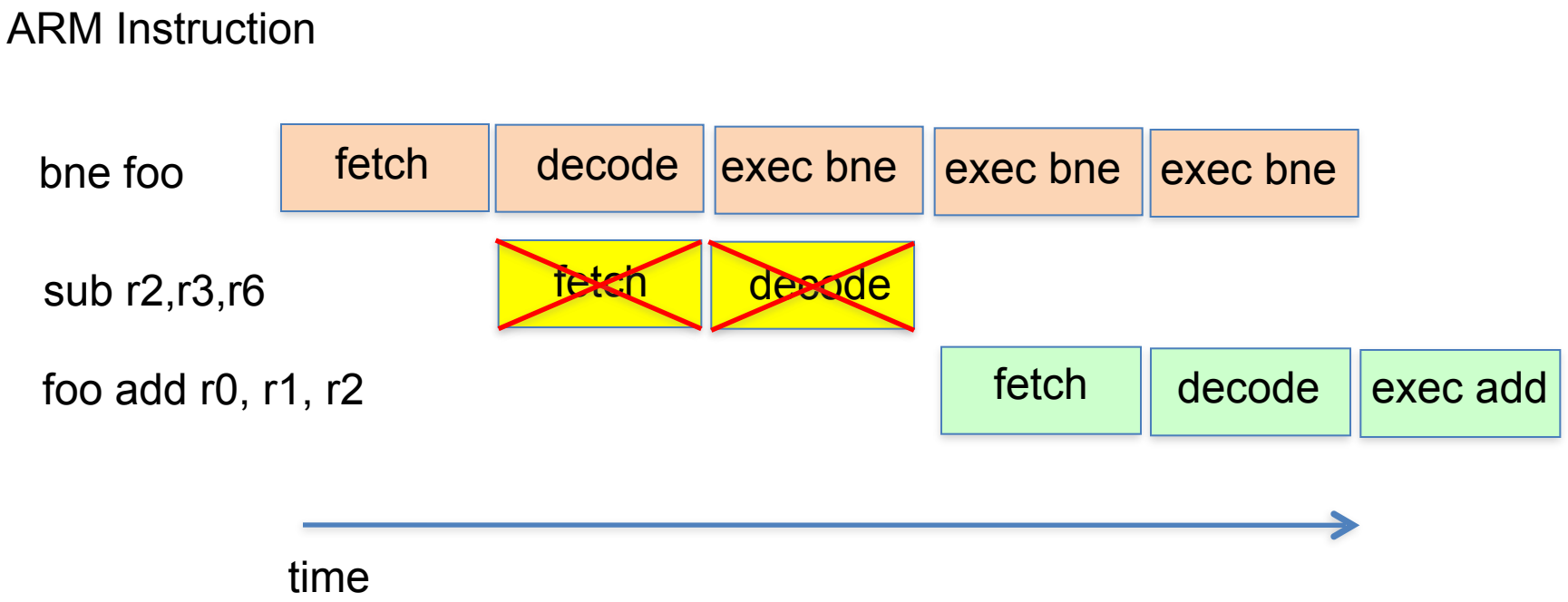
However, this ideal speedup cannot be reached in practice.

Data Stall

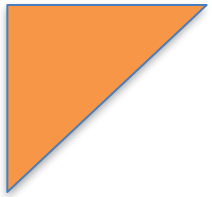


For the ldmia instruction two registers must be loaded and so two cycles required for the execution phase. Decode logic is also occupied so that the cpu can 'remember' the decoded instruction.

Control Stall



The decision whether to take the conditional branch bne is not made until the third clock cycle which computes the target branch address. If the branch is taken then the sub instruction should not be executed and the result of its fetch and decode are discarded.



Timing Improvement

If the delay through each stage of the pipeline is exactly the same then the pipeline is perfectly balanced and the speedup described on the previous slide can be (almost) achieved.

For the instructions given in the previous table it can be seen that the stages are not balanced and for some instructions some of the stages are not required but the pipelining scheme will still allocate time to them. All of these things reduce the potential speedup.

Also, this simple analysis has not considered added delays in the pipelining registers.